Cscape PID Quick Start Guide

PID Configuration and Tuning Quick Start Guide

1. PREFACE

The following guide acts as an introduction for users of PID (**P**roportional **I**ntegral **D**erivative) systems and controllers, and how to set up and configure a PID loop in Cscape with a Horner OCS. Information in this guide acts as supplemental information to the Cscape Help section covering PIDs. PID outputs can be used in a variety of applications, and all systems are different, so the information provided in the following guide will act as fundamental set up only.

The guide is written to start with the core principles of PID controllers for those who may not be as well versed in the process, then build into applying and tuning PID systems in Cscape.

The following topics will be covered:

- Introduction to PID
- PID Fundamentals
- Horner OCS Cscape Configuration
- Tuning

NOTE: Not all of the intricacies of PIDs will be explained here, but adequate information to be able to manage basic loops using Cscape.

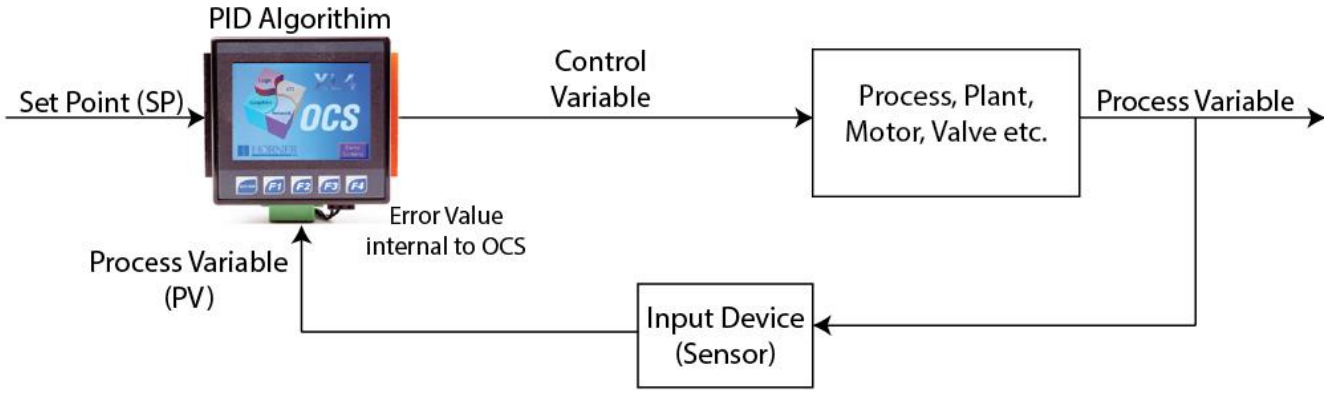
For a video on our PID Webinar **CLICK HERE**

2. INTRODUCTION

PID functions are outputs that change or control a physical property in a system. For new users, tuning PID systems can cause confusion and frustration. Examples of PID functions can be seen in systems that monitor and change pressures, temperatures, speeds, volume, etc. For instance, an HVAC thermostat provides an easy-to-explain illustration.

A household HVAC thermostat is set to a specific static temperature, a **Setpoint** in a PID system. The unit measures the ambient temperature of the home at any given time, generating a **Process Variable**, which is in a state of flux. The difference between the Set point and the Process Variable is referred to as the **Error** the PID system uses to calculate the **Output** back to the thermostat to keep the temperature at the set point. This process is usually in a continual, closed-loop state which would keep a consistent temperature (might not be energy efficient for HVAC to use PID loops).

Below is an illustration of a basic PID system:



3. PID Terms

Proportional Value – "K_P"

The Proportional Value **(K_P)** (sometimes called *Gain*) changes the Output in proportion to the current error value. The proportional value changes the output in a PID system the most, and it should be a starting point during the tuning process (which will be explained in more detail later). However, if the proportional value change is too high, it can result in an unstable system. Small proportional gain results in very small output change, which can be too small for a large input error, causing a less responsive (sensitive) controller.

Proportional only control will not settle at target value, but will retain a steady state error – a function of the proportional gain and the process gain.

The higher the proportional gain %, the larger the change in output.

Integral Value – "K_I"

The Integral Value **(K_I)** looks at an accumulation of past errors to calculate the next value change for the algorithm, thus can be illustrated as the change in value over a given time. The integral value accelerates the movement of the process towards set point and eliminates the residual steady-state error that occurs with proportional-only control. The process aims to reduce the steady state error to zero, creating a stable and effective PID system.

PI controllers are the most prevalent type of control, as they can normally stabilize at a set point without the need of the Derivative term. However, sometimes a derivative value is needed to stabilize the PID loop or increase responsiveness.

The higher the "repeats/sec" value of the Integral Term, the larger the resulting change. Be careful of overshoot.

Derivative Value – "K_d"

The Derivative Value **(K_d)** is proportional to the rate of change of the process variable. The Derivative term projects the current rate of change into the future by a set number of seconds, anticipating changes to the error. Adding the Derivative term to a PI loop can lead to a more responsive or faster loop because it allows for larger **P** and **I** gains. Derivative value is projected in seconds, meaning the number of seconds the value will project into the future to anticipate the rate of change.

However, if there is system noise in the PID loop, or oscillations, the **D** term can spike the other values and cause the PID algorithm loop to lose control, effectively rendering the loop unreliable.

Derivative values can lead to faster loop response, but be wary of system noise. Start without and add if needed.

Putting It All Together

Below is a basic chart that explains how the PID **generally** functions when **increased**, which will help in the following "Configuration" and "Tuning" sections:

Note: Each system is unique, so the following table explains general relationships only.

Value	Rise Time	Set Point Overshoot	Settling Time	Steady State Error
Proportional	Decreases	Increases	-	Decreases
Integral	Decreases	Increases	Increases	**Dissipates**
Derivative	-	Decreases	Decreases	-

4. CSCAPE CONFIGURATION

4.1 Choosing a PID function Block

There are two different PID block functions in Cscape: Independent and ISA. Both PID blocks use the set point and process variable to change a control variable using set algorithms. Both PID blocks will derive the same solution, but use different equations:

ISA:

$$CV_{OUT} = K_P*(Error + (K_I*Error*Dt/Ti) + (K_d*Derivative)) + CVBias.$$

Independent:

$$CV_{OUT} = K_P*Error + (K_I*Error*Dt/Ti) + (K_d*Derivative) + CVBias$$

For the majority of applications, either PID function could be used to tune a system. The Independent PID block is generally more common. In the ISA block, K_P is multiplied by the product of **(Error + (K_I*Error*Dt/Ti),** which would slightly change the way the values interact during the tuning process, but they both come to the same end result.

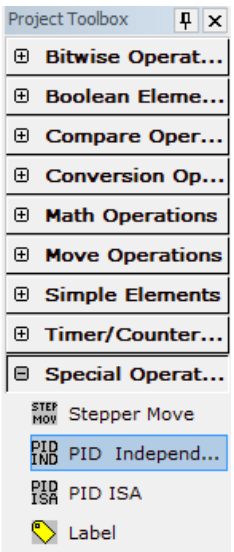
The simple mathematic equation above can cause confusion when going through the tuning process. The Cscape function block makes the tuning process easier by translating it into fields the user can fill in, including an option for auto-tuning.

4.2 Cscape Interface and Ladder Logic

PID functions can be set up a number of different ways. The following configuration shows a PID temperature controlled oven with heating coils driven by a relay that activates via PWM output. The example program is very simple, and may not illustrate every intricacy of PID functions.

Cscape can also utilize analog outputs to regulate a system.

Note: No two programs are the same. Use the following as a guide only.

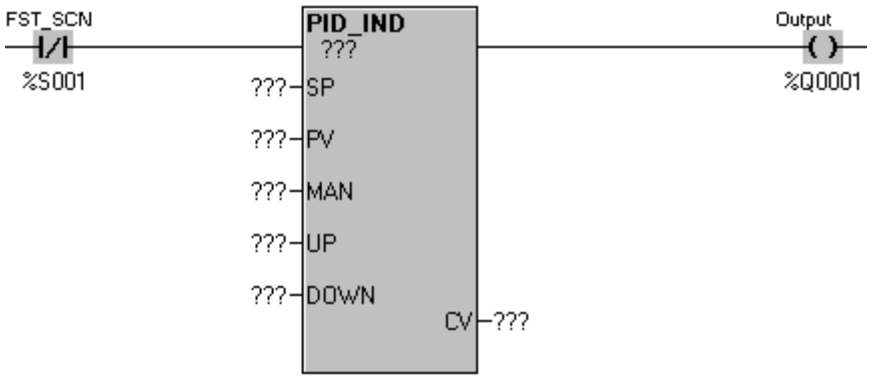


1) In this example, place a normally closed contact at the beginning of the rung. Double click it and change the address for %S001 for First Scan (**FST_SCN**).

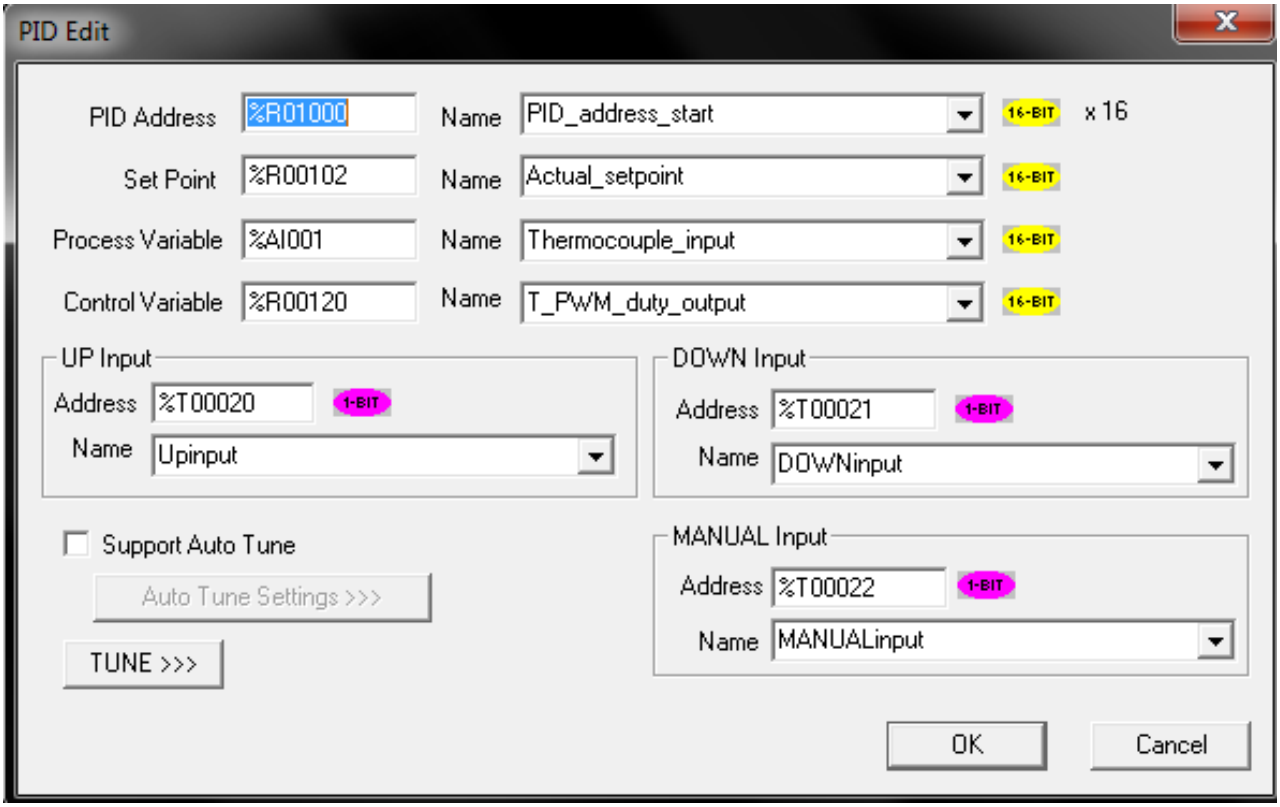
2) Open **Special Operations** in **Project Toolbox**. Select **PID Independent** (if desired **PID_ISA** can also be used) - see Figure 4.1.

3) Place the **PID_IND** block on the same rung.

4) Place normally open coil addressed to target output, which here = **%Q0001** (see Figure 4.2) If this output is analog, use %AQ0001. This example is using a relay to control PWM output, so %Q0001 is used



5) Configure registers in the PID block:



PID Address: A group of 16 consecutive registers that will be used to store the tune parameters to be used in the PID calculations (more details on specific registers later)

Set Point: Set point is the value that the PID is trying to achieve; the value the Process Variable (PV) aims to reach.

Process Variable: The current value from the process: the value that the PID is trying to change

Control Variable: Output of the PID. The value that will be used to control the process to bring the Process Variable towards the Set Point

MANUAL Input : Manual / Auto Boolean Switch. Enter a register address or select a named register that is the User-controlled Manual Input bit. This register is a Boolean (1-bit) register, typically %T.

UP input: Manual Mode up adjustment input. Enter a register address or select a named register that is the User-controlled UP Input bit. This register is a Boolean (1-bit) register, typically %T.

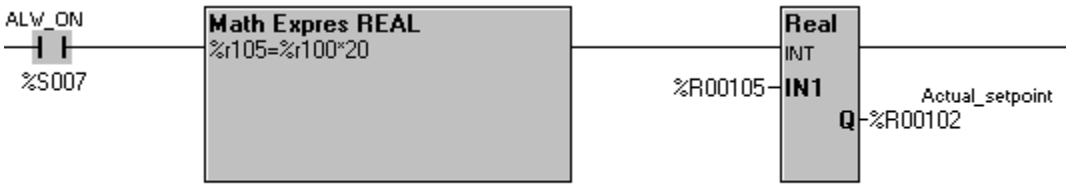
DOWN input: Manual Mode down adjustment. Enter a register address or select a named register that is the User-controlled DOWN Input bit. This register is a Boolean (1-bit) register, typically %T

Support AutoTune: Enables auto-tune

6) Input Degrees Celcius to INT Setpoint for PID block

Set a **Math Expres Real** block with a **%S007 ALW_ON** contact to define "%R105=%r100*20" Then, configure a **REAL to INT** block to convert the value to %R00102, which is the Actual_setpoint. (See below)

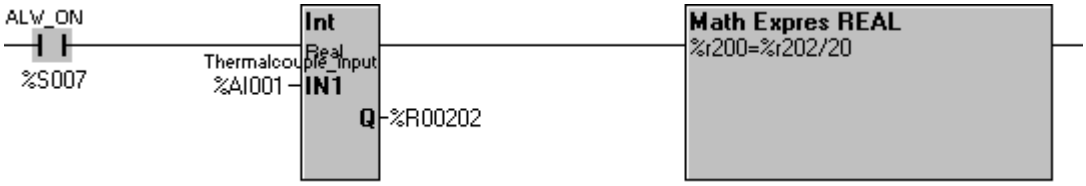
Note: In this example, %R00100 is set up as an editable field on a screen. The user sets a temperature in degrees Celsius, then this rung converts that number into an integer that the PID function block uses. %R00105 acts as a placeholder for the value as it goes through the conversion



7) Thermocouple Input Display in Degrees Celcius

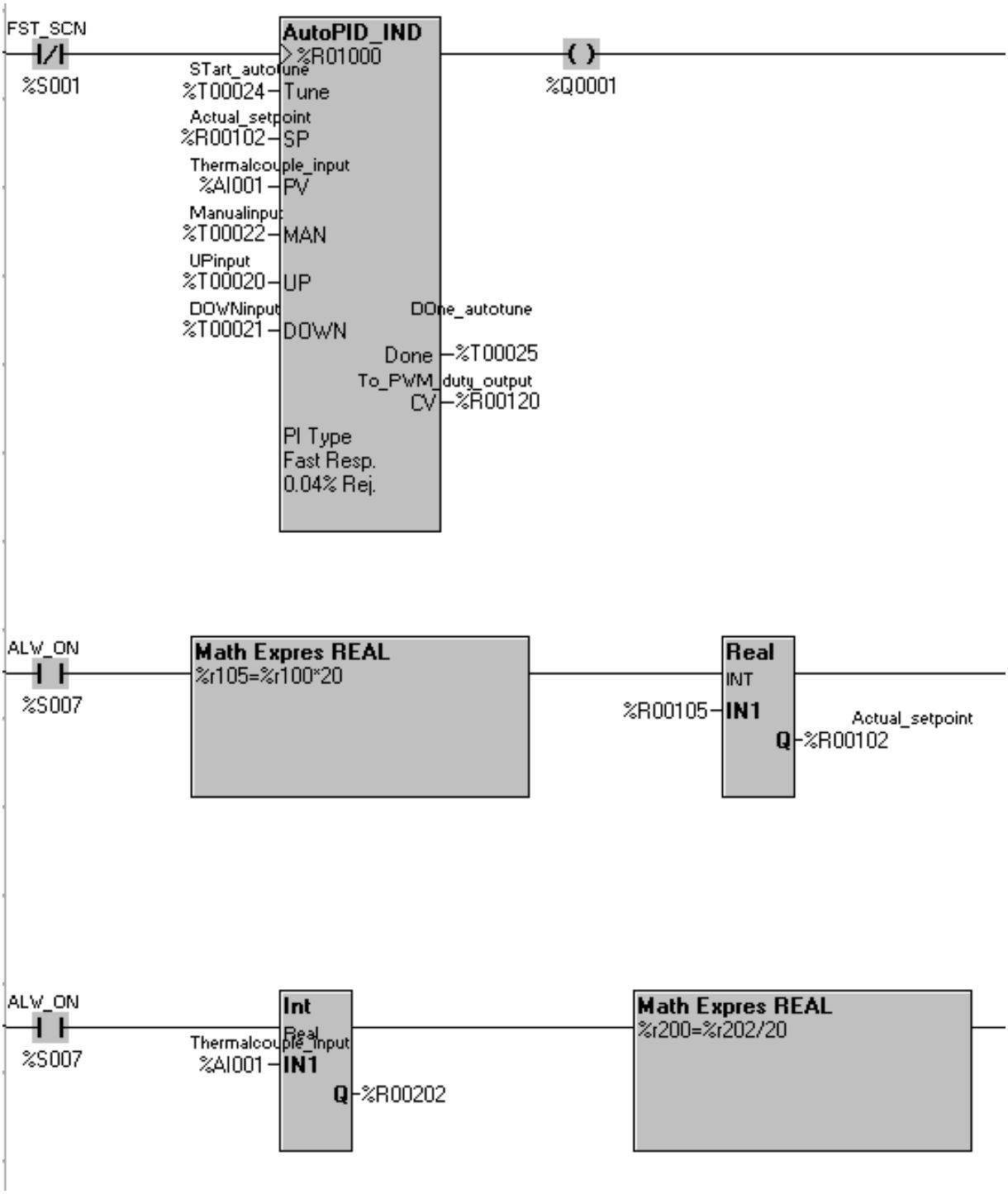
Place an **INT to REAL** function block with a **%S007 ALW_ON** contact to convert the integer input of the %AI001 thermocouple_input value to a real number at %R202. Then, configure a **Math Expres REAL** block on the rung to convert the %R202 thermocouple input into degrees Celsius for the display screen (%R200).

Note: In this example, %R200 will be used to display on the screen as the actual temperature in degrees Celsius. If Fahrenheit is preferred, simply replace the equation in the **Math Express REAL** box to convert to different measurement.



8) Create screens that have functionality to display or change certain fields of the program, such as changing the setpoint with %R102 above, or displaying the temperature in graph form of the temperature input from the thermocouple, like %R200 above. Also, either slide bars or editable fields can be created on the screens to make it easier and quicker to manually change PID terms for tuning purposes.

The following code shows all of the previous steps completed with Auto Tune set up on the PID block:



4.3 PWM Mode

For any application that requires a digital output (a heater that is controlled by either a relay or a digital output) PWM mode is an option. This will vary the power that passes through the block as a PWM signal. In order to control an output this way a coil must be place on the other side of a PID block.

For PWM mode, the **high clamp** and **low clamp** must be placed at **32000** and **-32000** respectively. -32000 is a duty cycle of 0% and 32000 is a duty cycle of 100%,

PWM cycle time: The PID cycle time determines how often the PID will turn on and off. It will switch up to once during the cycle time. If the cycle time is left at 0 the PID block will pass power whenever the CV value is positive, and not pass power when the CV value is negative, sometimes this is the desired mode of operation.

Selecting the right cycle time for the operation can be very important as it will directly affect how often your output will switch from ON to OFF. If you have any physical wear caused by turning the output from ON to OFF (like in a relay) this will help limit how often it changes.

4.4 Setpoints

When using PID blocks, all of the tunning parameters are stored in the command block. If %R registers are used, this data is retentive, meaning that it will stay in the registers after a power cycle.

If the PID program is going to be put into another controller, populate the registers associated with PID functions. This can be done via **Move** functions or through **setpoints**.

When using **Move** functions, information regarding what each register does is available in the PID control block. The setpoints can be obtained by using the **Save Setpoints** button in the PID tuning menu.

When using the setpoints, it is important to make sure that the setpoints populate the PID values. To do so, CsCape can be set up to download the setpoints into the controller.

Go to **Program→Download Options** and make sure the checkbox that says **Setpoint Table** is checked.

4.5 Registers

Both PID elements require an array of sixteen (16) WORD (16-bit) registers. These will typically be of type %R. This is called the *Reference Array*.

Each PID element must use a distinctly separate Reference Array, even if the values are identical to an existing PID element. There can be no overlapping of PID elements.

Registers at offset 0 through 9 must be configured before the PID element is used.

Register Offset	Parameter	Units	Range	Description
0	Sample Period	10 mS	0 to 65535	The shortest time in 10 mS Increments allowed between PID solutions
1	Dead Band +	PV counts	0 to 32000	Defines the Upper and Lower Dead Band limits in terms of PV counts. Both Should be set to 0 until the PID is tuned. A Dead Band might then be necessary to prevent small changes in CV values due to slight variations in error.
2	Dead Band -	PV counts	0 to -3200	
3	Proportional Gain (K_P) Percent	Percent	0 to 327.67%	Sets the Proportional Gain Factor in terms of percent. 100 sets unity gain (gain of 1).
4	Derivative Gain (K_d)	10 mS	0 to 327.67 seconds	Entered as a time with a resolution of 10 mS. In the PID equation this has the effect.
5	Integral Rate (K_I)	Repeats per 1000 seconds	0 to 32.76 repeats per second	Entered as a number of repeats per second. The integration rate.
6	CV Bias	CV Counts	-32000 to +32000	Number of CV counts added to the output before the rate and amplitude clamps.
7	CV Upper Clamp	CV Counts	-32000 to +32000	Number of CV counts that represent the highest and lowest value for CV. CV Upper Clamp must be more positive than CV Lower Clamp.
8	CV Lower Clamp	CV Counts	-32000 to +32000	
9	Minimum Slew Time	Seconds of full travel	0 to 32000 seconds to move 32000 CV counts	Determines how fast the CV value can change.
10	Config Word	N/A	N/A	**Internal Use – Do Not Modify this Value**
11	Manual Command	CV Counts	Tracks Auto in Auto Mode; sets the CV in Manual Mode	In the Automatic mode this register tracks the CV value. In Manual mode, this register contains the value that is output to the CV within the clamp and slew limits.
12	Internal SP	Used by OCS	N/A	Tracks SP in
13	Internal PV	Used by OCS	N/A	Tracks PV in
14	Internal CV	Used by OCS	N/A	Tracks CV out
15	Cycle time	Seconds	N/A	Cycle time for PWM in seconds

5. TUNING

5.1 Tune Parameters

It is important to know how the parameters in the PID block function. Below, definitions for each value are listed:

Parameter	Definition
Sample Period:	The time allowed between the PID calculations, this is how frequently the controller will calculate a new CV value.
Dead Band +/-	The dead band is used to prevent small changes in Control Variable values that may be caused by small changes in the process variable as it approaches the set point. This creates a range of values around the setpoint, if the process variable is within this range, then the control variable will not change.
Proportional Gain (K_p):	This is the gain value that is directly associated with the current error value, this is the K_P from the PID equation.
Derivative Gain (K_d):	For most applications Derivative Gain is unnecessary. This gain value is entered in with a resolution of 10mS, this value is used to make the control variable respond to sudden changes in the process variable
Integral Gain (K_i):	The integral gain is used to bring the average value of the process variable equal to the set point over time. It will accomplish this by using an integral to find the total error over a given period of time and make an adjustment to the Control variable. If the process variable is far away from the set point for a long period of time, this bias can build up. Once it has been built up it may take a while to change, this is something to watch out for in processes that may stay a long time at one set point and then make a large change to another.
CV Bias:	This is a value that is added to the Control variable after the calculation has taken place. The control variable will be calculated, the bias will be added in, and then any slew or amplitude limits will take effect.
CV upper and lower clamp:	The control variable cannot be higher than the upper clamp and it can't be lower than the lower clamp. This will make sure that your control variable stays within a defined range.
Minimum Slew time:	This will limit how fast the Control variable will change. The units are number of seconds it takes to move from 0 to 32000, so a slew rate of 64 will let the control variable change no more than 500 counts per second.
Cycle Time (PWM mode only):	This is the duty cycle for the PWM output functionality in seconds, this can be used to limit how frequently the output will switch from ON to OFF. If this is left at 0, the block will pass power if the CV is 0 or greater, and not pass power if the CV is less than 0.

5.2 Manual Tuning

Manual Tuning K_P

For many applications a quick manual tuning is all that will be necessary, most applications will use a K_P value and a small K_I value. When setting the initial values, it can be helpful to view the PID in terms of mathematics.

Example: For a temperature control application, in order to decide where the K_P value should start at think in terms of temperature:

- For example, for output to be at maximum (32,000) when my temperature is at least 20°below the setpoint
- Convert 20° to raw counts from an analogue input, if the input has a resolution of 0.05° then this gives a raw count of 400

- The equation with just the constant gain is $CV_{OUT} = K_P$*Error, since the CV_{OUT} and the Error are known, fill in the equation 32000 = K_P*400
  32000/400 = K_P
  K_P = 80
- Now we have a starting point for K_P , it is important to test this out in the actual application

Manual Tuning K_I

For the majority of applications the K_I value does not need to be very large. There are several things to consider when determining what kind of K_I value your application should have.

The way the integral value works is it takes the integral of the error over a period of time and adds it to an "invisible" CV bias. This will continue to increase or decrease until your PV averages out to be your set point.

If the process is slow, watch out for a large value in this "invisible" bias. If there is a change in the setpoint, this value may not change very quickly and can make the response time of your system significantly slower.

The Math:

Integral_Bias = Integral_Bias + Ki*.001(dt, time elapsed since last measurement in seconds)*error

For simplicity, look at the increase over a second and assume that for the duration of the second the process variable remains unchanged.

Example(s): Set Point = 1000, the process variable = 800

Ki	Change in Control variable per second
100	20
250	50
600	120
1000	200

The big danger with integral is that it is always active. If a process begins with a set point of 5000, the process variable starts at 2000, and the process variable takes 3 minutes to reach the set point, in that time with a Ki value of 100 in that time the Bias will be at a value of 27000 counts. This will likely lead to significant overshoot for the process variable.

5.3 Auto-Tuning the PID Block

Settings for the PID Auto Tune	
Start Auto Tune Address	A 1-bit address that will be set high during the auto-tune process
Auto Tune Done Address	A 1-bit address that will be set high after the auto-tune process is complete
Auto Tune Type	A drop down menu to select the type of tuning that will be done. The three letters each stand for a type of gain Proportional, Integral, Derivative . All three can be selected, Proportional and Integral , or just Proportional. (**PID**, **PI**, or **P** respectively)
Noise Filtering	The noise filtering determines how much above and below the set point the Auto Tune process must go to obtain its measurement. This will prevent noise in the measurement from signaling that the auto Tune has reached the desired set point pre-maturely. The percentage is based on full scale (32000 counts) and ranges from 0.04% (~13 counts) to %5 (1600 counts)
Controller Response	Determines the "speed" of the response that is auto tuned. **Fast** will produce some overshoot but arrive at the set point faster, **Medium** will produce slight overshoot and arrive at the set point a little slower, **Slow** will produce no overshoot, **Very Slow** may need to be selected for a process that is "outside the optimum range for Zeigler-Nichols rules" – very sensitive systems
Tune at 2/3 Setpoint	Allows the auto tuning experiment to change the output based on 2/3 the set point. Use this option when it is not desired for the process to travel above the set point during the auto tuning experiment.

Prior to Auto tuning make sure to set up the following values in the PID block:

- Set point
- Sample Period
- Upper and lower Clamp
- Error Term,
- Output Polarity

Once all of the settings are in place, set the Start Auto Tune bit High (either in **Data Watch** or in the code)

The system will take control and cause the Process Variable to go above and below the setpoint several times, after it has taken all of the measurements it needs the **AutoTune_Done** bit will go high and the PID block will go back to its last state with the "optimized" Proportional, Derivative, and Integral values in place.

In CsCape open up the PID block to look at the values, either save them as new set points or make note of them for future use.

6. TECHNCIAL SUPPORT

NORTH AMERICA

- Telephone: 317-916-4274 Fax: 317-639-4279
- http://www.heapg.com Email: techsppt@heapg.com

EUROPE

- Telephone: +353-21-4321266 Fax: +353-21-4321826
- http://www.horner-apg.com Email: tech.support@horner-apg.com